An Infrastructure Modeling Approach for Multi-Cloud Provisioning

Julio Sandobalin *julio.sandobalin@epn.edu.ec*
Departamento de Informática y Ciencias de la Computación, Escuela Politécnica Nacional Quito, Ecuador

Emilio Insfran *einsfran@dsic.upv.es*
Departamento de Sistemas Informáticos y Computación, Universitat Politècnica de València Valencia, Spain

Silvia Abrahao *sabrahao@dsic.upv.es*
Departamento de Sistemas Informáticos y Computación, Universitat Politècnica de València Valencia, Spain

Abstract

Cloud Computing has become the primary model of pay-per-use used by practitioners and researchers to obtain an infrastructure in a short time. DevOps uses the Infrastructure as Code approach to infrastructure automation based on software development practices. Moreover, the DevOps community provides different tools to orchestrate the infrastructure provisioning in a particular cloud provider. However, the traditional method of using a single cloud provider has several limitations regarding privacy, security, performance, geography reach, and vendor lock-in. To mitigate these issues industry and academia are implementing multiple clouds (i.e., multi-cloud). In previous work, we have introduced ARGON, which is an infrastructure modeling tool for cloud provisioning that leverages the model-driven engineering (MDE) to provide a uniform, cohesive, and seamless process to support the DevOps concept. In this paper, we present an extension of ARGON to support the multi-cloud infrastructure provisioning and propose a flexible migration process among cloud.
Keywords: Infrastructure Provisioning, Infrastructure as Code, Cloud Computing, Multi-Cloud, DevOps, Model-Driven Engineering.

1. Introduction

In many of today's enterprises, one of the most important challenges is how to deliver a new idea or software artifact to customers as fast as possible. To face this issue, practitioners and researchers are using a new trend called DevOps (Development & Operations) [6], which is promoting continuous collaborations between developers and operation staff through a set of principles, practices, and tools to improve the software delivery time. The cornerstone of DevOps is the Infrastructure as Code [9], which is an approach to infrastructure automation based on software development practices that emphasize the use of consistent and repeatable routines for infrastructure provisioning.

On the other hand, cloud computing has become the primary model of pay-per-use used by practitioners and researchers to obtain an infrastructure in a short time. According to Brikman [1] the use of DevOps on cloud-based processes is causing some shifts, such as:

- Instead of managing data centers, many companies are moving to the Cloud, taking advantage of services such as Amazon Web Services, Microsoft Azure, and Google Computing Engine.
- Instead of investing heavily in hardware, many operations teams are spending all their time working on software, using DevOps community tools such as Chef, Puppet, Terraform, and Docker.

- Instead of racking servers and plugging in network cables, many sysadmins are writing code.

Cloud-based processes that use DevOps apply the Infrastructure as Code approach and leverage DevOps community tools for cloud infrastructure provisioning tasks. In this scenario, developers and operation staff focus their efforts on working on software, i.e., writing code to define the cloud infrastructure using scripts. As a result, it is possible to write and execute code (i.e., script) to define, deploy, and update the cloud infrastructure.

Every team, department, or software application of a company have their own requirements in terms of privacy, security, performance, or geography reach. Similarly, different cloud providers offer different characteristics. For this reason, companies are starting to use multiple clouds to satisfy their needs to reach flexibility and agility required by the market. In this context, industry, and academia have begun to use the term multi-cloud to refers to the use of multiple clouds without relying on any interoperability functionalities implemented by the providers [5].

Despite the enormous contribution of DevOps community to bridging the gap regarding orchestration of infrastructure provisioning for multi-cloud approaches, there still exist issues to solve, such as:

- Manage script languages of different DevOps community tools for infrastructure provisioning is a time-consuming and error-prone activity.
- Cloud providers do not offer the same type of infrastructure. Therefore, it is necessary to define a custom script for infrastructure provisioning for every cloud provider.
- Lack of portability between cloud providers and vendor-lock-in are issues that should be avoided. DevOps community tools still do not provide a flexible process to support the migration among cloud providers.

To mitigate the issues above mentioned, in a previous work we have presented ARGON [12], which is an infrastructure modeling tool for cloud provisioning. ARGON aims to abstract the complexity to work with different cloud providers through a Domain-Specific Language. ARGON allows modeling a generic infrastructure model and generates the corresponding scripts to manage the different DevOps community tools for cloud infrastructure provisioning (henceforth, DevOps provisioning tools). In this paper, we present an extension of ARGON which leverages the model-driven engineering (MDE) for supporting the multi-cloud infrastructure provisioning. The contributions of this work are: (i) a multi-cloud infrastructure modeling approach, and (ii) a flexible migration process among cloud providers. To demonstrate the feasibility of our proposal we use cloud providers such as Amazon Web Services and Microsoft Azure.

The remainder of this paper is structured as follows. Section 2 discusses related works and identifies the needs of multi-cloud infrastructure provisioning. Section 3 presents a brief introduction to ARGON. Section 4 presents our approach of multi-cloud infrastructure provisioning and the flexible migration process among cloud providers. Section 5 presents a case study which demonstrates the feasibility of our proposal. Finally, Section 6 presents our conclusion.

2. Related Work

In recent years, there has been much interest, and many approaches and strategies emerged to support cloud infrastructure provisioning. For instance, Amazon Web Services provides infrastructure modeling tools such as CloudFormation [16] and OpsWorks [17]. CloudFormation promotes a common language for describing and provisioning all the infrastructure resources. OpsWorks is a configuration management service that provides managed instances of DevOps provisioning tools such as Chef and Puppet.

CloudMF [4] is a Cloud Modeling Framework which proposes a Domain-Specific Language (DSL) for specifying the provisioning and deployment of multi-cloud applications. The Cloud Provider-Independent Model (CPIM) defines the provisioning and deployment in an agnostic way. The Cloud Provider-Specific Model (CPSM) uses a model@run-time engine

to requests cloud providers for a list of available resources and use them to refine the CPIM into a CPSM.

MUSA [2] is a framework which provides a DevOps approach to develop multi-cloud applications with desired security Service Level Agreements (SLAs). The MUSA Modeler Tool relies on a specific modeling language based on CAMEL [11] to describe the application architecture and the deployment requirements. The MUSA Risk Assessment Tool carries out a risk assessment process to identify the security Service Level Objectives (SLOs) required by the multi-cloud application components. Finally, MUSA generates the security Service Level Agreement (SLA) templates for the components by means of the MUSA SLA Generator tool.

MODAClouds [10] is a European project which delivers an advanced software engineering model-driven approach and an integrated development environment to support systems developers in building and deploying applications towards multi-clouds. MODAClouds allows defining the Quality of Service (QoS) requirements at the Cloud Independent Model level (CIM). Then, cloud-specific aspects are introduced at the Cloud-Provider Independent Model level (CPIM). Finally, the Cloud-Provider Specific Model level (CPSM) specifies a particular provider and service for the application, run precise QoS analyses and generate proper deployment, monitoring, and self-adaptation scripts to support the runtime phases.

MORE [3] is a Topology and Orchestration Specification for Cloud Application which allows modeling nodes (virtual or physical machines) and orchestrates the deployment of Cloud-based applications. TOSCA uses DevOps community tools such as Chef and Juju for infrastructure provisioning and cloud-based applications implementation.

The research works mentioned above focus their efforts on providing support for the modeling and deployment of multi-cloud applications as well as to manage both the Provider-Independent Model (PIM) and the Provider-Specific Model (PSM). In contrast, ARGON provides a Domain-Specific Language (DSL) for modeling the cloud infrastructure provisioning and a process for managing the complexity of handling the PIM and the PSM. Moreover, ARGON automatically generates scripts of infrastructure provisioning for different DevOps provisioning tools.

3. ARGON

ARGON (An infRastructure modellinG tool for clOud provisioNing) [12] is a tool that leverages Model-Driven Engineering and supports the DevOps ideas.

3.1. Modeling the Cloud Infrastructure Provisioning

There exist several cloud providers that provide different types of infrastructure, for instance, Amazon Web Services and Microsoft Azure. To mitigate the complexity of working with different providers and tools, we have developed a Domain-Specific Language (DSL) for modeling a generic infrastructure model.

Abstract Syntax

ARGON defines a generic Infrastructure Metamodel [12], which abstracts de cloud capacities such as computing, storage, networking, and elasticity. **Fig. 1a** shows an excerpt of the Infrastructure Metamodel.

- Computing capacity allows modeling Virtual Machines with its Security Groups. A Security Group performs like a firewall. Each Security Group enables connections from/to Virtual Machines through Inbound and Outbound rules. A Static IP address can be assigned to a Virtual Machine. A Load Balancer allows distributing the workload among Virtual

Machines. A Listener checks connection requests to the Load Balancer. A Health Check validates that Virtual Machines attached to the Load Balancer are available.

- Storage capacity allows modeling Databases and File servers.
- Elasticity capacity allows modeling a Launch Configuration in which characteristics of a Virtual Machine are specified. An Auto Scaling Group determines the minimum and the maximum number of Virtual Machines to be created. Creation or elimination of Virtual Machines is done based on a Scaling Policy in which is executed an Alarm that monitors a metric at a time frame.
- Networking capacity allows modeling the associations among metaclasses.

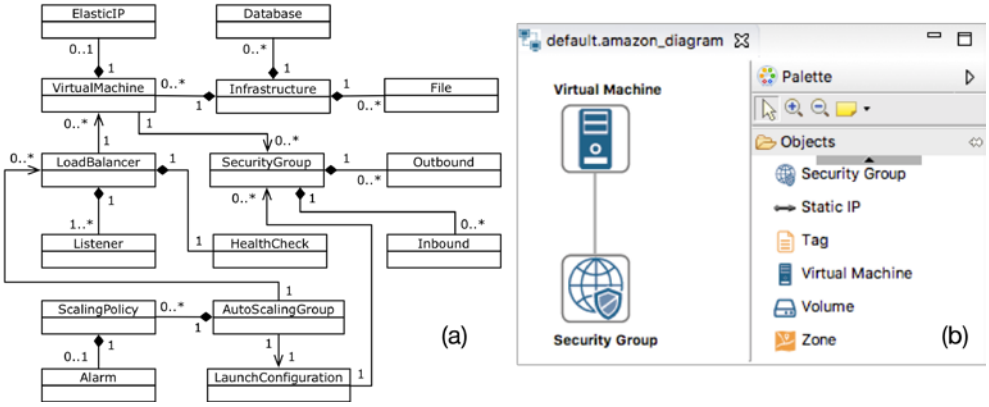

Fig. 1. (a) Infrastructure Metamodel. (b) ARGON: graphical notation.

Concrete Syntax

We use Eugenia [8] to render the Infrastructure Metamodel in a modeling editor such as Graphical Modeling Framework [18] (GMF).

Eugenia is a tool which facilitates to generate the models needed to implement a GMF editor in Eclipse from a single annotated Ecore metamodel, i.e., Infrastructure Metamodel. Eugenia allows automatically create the models required to accomplish the concrete syntax (i.e., graphical notation) in GMF. **Fig. 1b** depicts an Infrastructure Model in which a Virtual Machine is connected to a Security Group. Moreover, **Fig. 1b** shows the infrastructure elements palette which should be used to model an Infrastructure Model.

4. Multi-Cloud Infrastructure Modeling

Each cloud provider has a different type of infrastructure making more difficult to define a generic infrastructure provisioning solution. In this context, we take advantage that each cloud provider specifies their infrastructure following the same general cloud capacities (i.e., computing, storage, networking, and elasticity). Thus, we abstract these capacities at a higher-level of abstraction to define a generic infrastructure model. In addition, to provide support to a multi-cloud approach is also necessary to specify all the features of each cloud provider. For this reason, we have defined a platform-specific metamodel for each cloud providers such as Amazon Web Servers and Microsoft Azure.

4.1. Multi-Cloud Architecture

ARGON follows the Model-Driven Architecture (MDA) principles. According to MDA, we define several layers to abstract the complexity of working with different cloud providers and also different DevOps provisioning tools. **Fig. 2** introduces the ARGON architecture at different abstraction layers. Following we describe each layer in more detail.

Requirements

It is the most abstract layer of MDA, which represents the context, requirements, and purpose of the solution without any binding to cloud providers. **Fig. 2a** shows the layer in which the criteria to get a cloud, and multi-cloud infrastructure provisioning approaches are captured. As a result, we have defined the structure and behavior of the respective platform-independent model (which is unique and generic) and the platform-specific models (one for each cloud platform) by defining their metamodels.

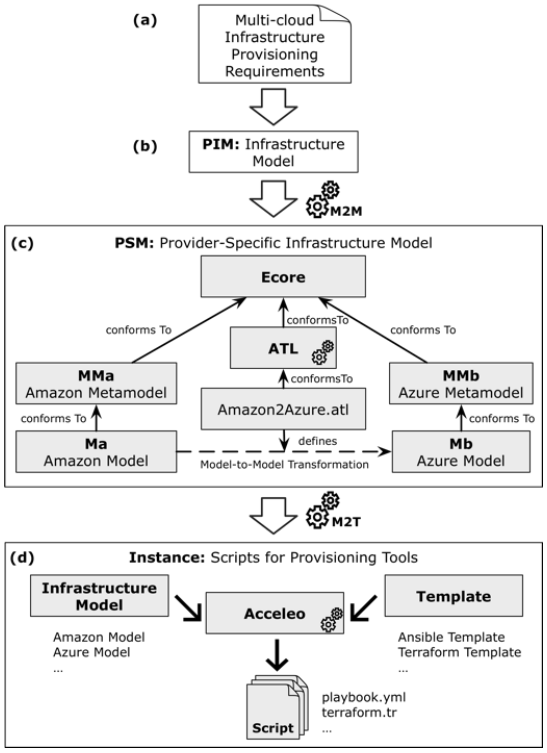

Fig. 2. Multi-Cloud Architecture

Platform-Independent Model (PIM)

In this layer, we define the behavior and structure of a generic solution to model the cloud infrastructure by means of an Infrastructure Metamodel (see **Fig 1a**) regardless of the cloud providers. **Fig. 2b** represents a generic Infrastructure Model which abstract the cloud capacities (i.e., computing, storage, networking, and elasticity) specifying the requirements using the ARGON's Domain-Specific Language (see Section 3). The Infrastructure Metamodel was defined using the metamodeling language Ecore [13].

Platform-Specific Model (PSM)

In this layer, we have two cases: (i) PIM to PSM, where an Infrastructure Model is transformed into an Amazon Web Services Model (based on an Amazon Metamodel) or into a Microsoft Azure Model (based on an Azure Metamodel); (ii) PSM to PSM, where an Amazon Model is transformed into an Azure Model (see **Fig. 2c**). Both Amazon Metamodel and Azure Metamodel are modeled using the metamodeling language Ecore. The transformation language for performing the Model-to-Model (M2M) transformations is ATL [7]. To perform the transformation from an Amazon Model to an Azure Model, we define a transformation module (e.g., Amazon2Azure.atl) that contains the transformation mapping rules which represent the correspondences among concepts between the Amazon Metamodel and Azure Metamodel. The M2M transformation process (see **Fig. 2c**) begins when the ATL transformation engine takes an Amazon Model (i.e., source model) and apply a set of transformation rules to obtain the Azure Model (i.e., target model).

It is worth noting that we can interchange the source and target models, meaning that an Azure model can be transformed into an Amazon model given that the correspondences among concepts are bidirectional. In addition, another cloud provider model can be used by defining the respective metamodel and transformation module, e.g., for Google Computing Engine, Rackspace, etc.

Scripts for Provisioning Tools (Instances)

In this layer, we generate the corresponding scripts for a particular DevOps provisioning tool (i.e., Ansible, Terraform, Chef, Puppet, etc.) by means of Model-to-Text (M2T) transformations. The source for this transformation process is a PSM (e.g., Amazon model or Azure model) and the output is an Instance layer (see **Fig. 2**). The scripts are used to manage the orchestration of the infrastructure provisioning in the cloud providers. Each DevOps provisioning tool uses a specific type of script, for instance, the Ansible tool uses a script called playbook, the Puppet tool uses a script called manifest, etc. Moreover, the DevOps provisioning tools use different script languages to define instructions for infrastructure provisioning, for instance, the Ansible tool uses the YAML language, the Terraform tool uses the HashiCorp Configuration Language (HCL), etc.

We abstract the specific features of each script language to define transformation rules in templates, which allow us to generate the specific scripts. These templates are defined according to the Acceleo [15] tool. The M2T transformation process (see **Fig. 2d**) takes as input a cloud provider-specific model (i.e., Amazon model or Azure model) and apply a set of transformation rules to generate scripts, as output, for a DevOps provisioning tool selected. As a result, thanks to this layered architecture, we can generate scripts for a variety of DevOps provisioning tools.

4.2. Cloud providers metamodeling

In order to achieve the multi-cloud modeling, it is necessary to abstract the cloud capacities (i.e., computing, storage, networking, and elasticity) of each cloud provider. To show the feasibility of our proposal, we have abstracted the cloud capacities of providers such as Amazon Web Services and Microsoft Azure.

Amazon Web Services Metamodel

Fig. 3 shows the Amazon Web Services Metamodel, which has a central metaclass called Service. The Service metaclass has the role of a container for infrastructure elements, and it allows setting the region in which the infrastructure will be deployed in Amazon Web Services. Element is an abstract metaclass that has fundamental attributes (e.g., name) which will be inherited by the rest of infrastructure elements. Tag is a metaclass to put data in the key-value style. The rest of infrastructure elements are explained according to cloud capacities.

In terms of computing capacity, we can model Virtual Machines with its Security Groups. A Security Group performs like a firewall. Each Security Group allows connections from/to Virtual Machines through Inbound and Outbound rules. An Elastic IP address can be assigned to a Virtual Machine. A Volume which is like an external disk can be attached to a Virtual Machine. A Load Balancer allows distributing the workload among Virtual Machines. A Listener checks the connection requests to the Load Balancer. A Heath Check validates that Virtual Machines attached to Load Balancer are available. A Zone allows distributing Virtual Machines across multiple availability zones, and one instance fails. Cloud Profile is an abstract metaclass which provides general attributes corresponding to Virtual Machines of others cloud providers. However, an Azure Profile metaclass provides specific attributes corresponding to Virtual Machines from Microsoft Azure. To provide support for others cloud providers should be added new metaclasses which inherit from Cloud Profile.

In terms of storage capacity, we can model Databases and Buckets.

In terms of elasticity capacity, we can model a Launch Configuration in which characteristics of a Virtual Machine are specified. An Auto Scaling Group determines the minimum and the maximum number of Virtual Machines to be created. Creation or elimination of Virtual Machines is done based on a Scaling Policy in which is executed an Alarm that monitors a metric at a time frame.

Finally, networking capacity is modeled through associations among metaclasses.

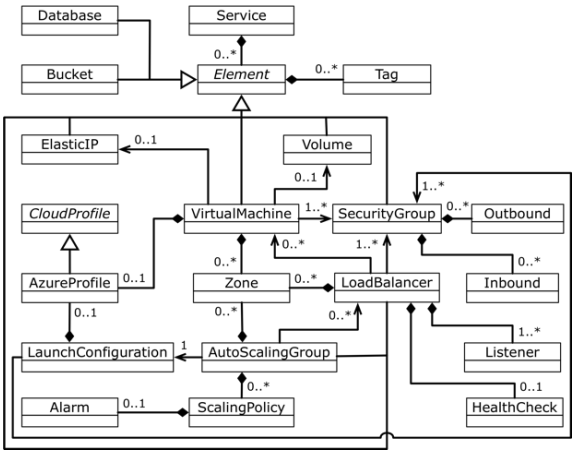

Fig. 3. Amazon Web Services Metamodel

Fig. 4 shows the Microsoft Azure Metamodel, which has a central metaclass called Resource Group. The Resource Group metaclass has the role of a container for infrastructure elements, and it allows setting the location in which the infrastructure will be deployed in Microsoft Azure. Moreover, Resource Group is one more element of the infrastructure. Element is an abstract metaclass that has fundamental attributes (e.g., name) which will be inherited by the rest of infrastructure elements. Tag is a metaclass to put data in the key-value style. The rest of infrastructure elements are explained according to cloud capacities.

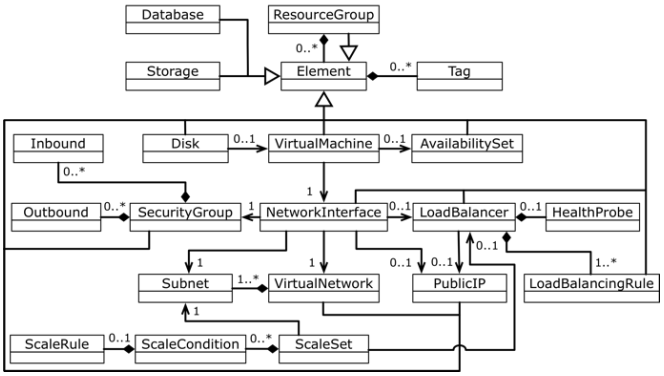

Fig. 4. Microsoft Azure Metamodel.

Regarding computing capacity, we can model Virtual Machines with its Security Groups. A Security Group performs like a firewall. Each Security Group allows connections from/to Virtual Machines through Inbound and Outbound rules. A Public IP address can be assigned to a Virtual Machine. A Disk which is like an external disk can be attached to a Virtual Machine. A Load Balancer allows distributing the workload among Virtual Machines. A Load Balancing Rule checks the connection requests to the Load Balancer. A Heath Probe validates that Virtual Machines attached to Load Balancer are available. An Availability Set ensures that the Virtual Machines are distributed across multiple isolates hardware nodes in a cluster.

Regarding storage capacity, we can model Databases and Storages.

Regarding elasticity capacity, we can model a Scale Set in which characteristics of a Virtual Machine are specified and should be set the minimum and the maximum number of Virtual

Machines to be created. Creation or elimination of Virtual Machines is done based on a Scale Condition in which is executed a Scale Rule that monitors a metric at a time frame.

Regarding networking capacity, we should model a Virtual Network which enables Azure resources to communicate with each other in a network securely. A Virtual Network can be segmented into multiples Subnets. Finally, a Network Interface allows a Virtual Machine to interact with others resources in a Virtual Network.

Multi-cloud Infrastructure Provisioning Modeling

In order to illustrate the feasibility of our proposal, first, we have modeled an Infrastructure Model (see **Fig. 5**) of a Virtual Machine. Next, the Infrastructure Model (PIM) will be transformed into an Amazon Model (PSM). Finally, the Amazon Model will be migrated toward an Azure Model through Model-to-Model (M2M) transformations.

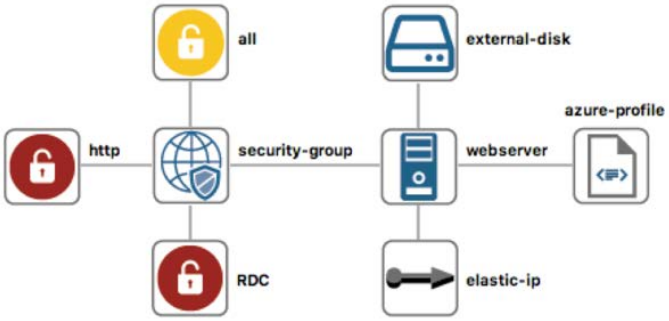

Fig. 5. Infrastructure Model of a virtual machine.

Fig. 5 shows an Infrastructure Model of a virtual machine modeled by ARGON. In this case, by using M2M transformations, the Infrastructure Model is specified as an Amazon Model (PIM to PSM, see **Fig. 2**). The Infrastructure Model has a *webserver* connected to a *security-group*, which performs like a firewall. The *security-group* has an outbound rule (*all*) which enables all outgoing connections of the *webserver*. Moreover, the *security-group* has two inbound rules which enable the ingoing connections to *HTTP* protocol and Remote Desktop Connection (*RDC*) for a Windows Server. The webserver has connected an *external-disk* and an *elastic-ip*.

The webserver (see **Fig. 5**) has settings corresponding to a virtual machine from Amazon Web Services, however, to add features for a virtual machine from Azure platform (such as publisher, offer, and SKU) is necessary to use an *azure-profile* element.

Fig. 6 shows specifications of M2M transformation to migrate a virtual machine from Amazon Web Services to Microsoft Azure. Following are explained changes:

- *Service2Network* and *Service2ResourceGroup* specify that for each *Service* container in the Amazon Model should be created a *Resource Group* container and a *Virtual Network* with its *Subnet* in the Azure Model.
- *Tag2Tag* specifies that for each *Tag* found in an element in the Amazon Model should create the same *Tag* in the corresponding element in the Azure Model.
- *SecurityGroup2SecurityGroup* specifies that for each *Security Group* with its *Inbound* rules and *Outbound* rules in the Amazon Model should be created the corresponding *Security Group* with its *Inbound* rules and *Outbound* rules in the Azure Model.
- *Volume2Disk* specifies that for each *Volume* attached to a virtual machine in the Amazon Model should be created the corresponding *Disk* connected to the virtual machine in the Azure Model.
- *VirtualMachine2VirtualMachine* specifies that for each *Virtual Machine* in the Amazon Model should be created the corresponding *Virtual Machine* with its *Network Interface* in the Azure Model.

- *ElasticIP2PublicIP* specifies that for each *Elastic IP* assigned to a virtual machine in the Amazon Model should create the corresponding *Public IP* assigned to the virtual machine in the Azure Model.

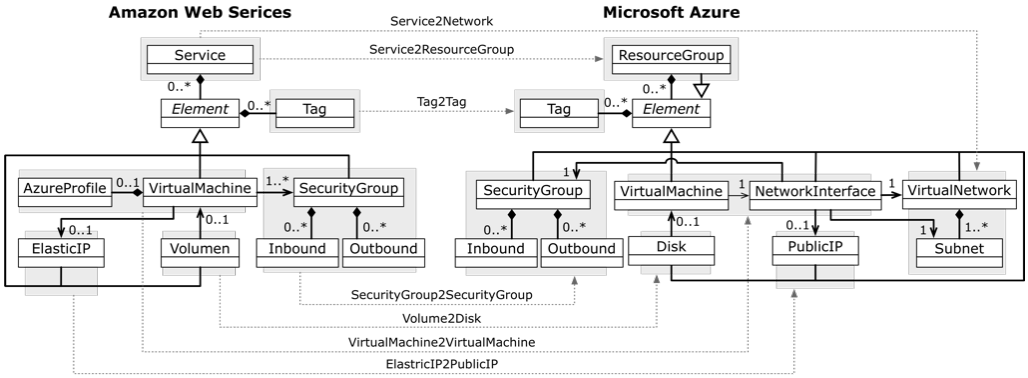

Fig. 6. Specification of M2M transformations for a Virtual Machine.

Fig. 7 presents in a hierarchical tree view the matching among infrastructure elements of the Amazon Model (see **Fig. 7a**) and the Azure Model (see **Fig. 7b**). In this context, the infrastructure Service called *virtual-machine* is a container in the Amazon Model (group A), which is matched to a Resource Group that is a container and also a Virtual Network with its Subnet in the Azure Model (group A'). The Virtual Machine called *webserver*, and its Azure Profile in the Amazon Model (group B) are matching to Virtual Machine and its Network Interface in the Azure Model (group B'). The rest of the infrastructure elements correspond one by one in the matching between the Amazon Model and the Azure Model.

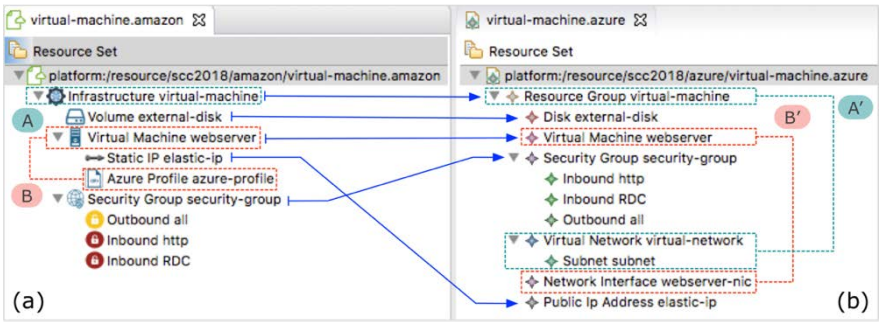

Fig. 7. Matching of elements between virtual machine models.

5. Case Study Description

In order to illustrate our approach, we refer to the case of a small company called MODAFIN (adapted and extended from [9]). MODAFIN is specialized in IT applications for financial services. Its main product line is a proprietary solution for stock market operations, cash administration, and lending management. MODAFIN most profitable activities are software customization and life-cycle management for this product line. Customization involves the development of custom modules to accommodate new functional requirements.

The consultancy team has been working for a long time on a software application, which will be deployed on Linux Servers in Amazon Web Services. However, the consultancy team is worried about issues such as vendor lock-in, security, and application performance. For this reason, the consultancy team has decided to use multiple clouds (i.e., multi-cloud) to deploy the software application. Consequently, Microsoft Azure has been selected as an alternative platform to deploy the application.

The infrastructure requirements to deploy the software application are the following:

Req. 1: A load balancer should distribute the workload between two virtual machines. The load balancer should check the connection requests to port 80 and validate that all virtual machine attached are available.

Req. 2: A firewall should enable ingoing connections only for the SSH protocol (port 22) and the HTTP protocol (port 80). The SSH protocol allows establishing a connection among virtual machines. The HTTP protocol allows establishing connections with the software application. Moreover, all outgoing connections should be enabled.

Req. 3: For security reason, connections to servers that are outside of cloud provider should not be enabled to virtual machines attached to the load balancer. In this case, a jumpbox server should be created to secure management of the virtual machines connected to the load balancer. The jumpbox server enables connections between servers that are outside the cloud provider and virtual machines attached to the load balancer.

The multi-cloud requirements to deploy the infrastructure are the following:

Req. 4: The proposed solution should support a deployment process in multi-cloud. Cloud providers selected are Amazon Web Services and Microsoft Azure.

Req. 5: The proposed solution should provide a flexible migrate process for cloud providers. In this case from Amazon Web Service to Microsoft Azure.

Following, we explain how to provide a solution for the infrastructure requirements:

Solution to Req. 1: Fig. 8 shows an Infrastructure Model, which has a *load-balancer* that distribute the workload between two *virtual machines*. **Fig. 9a** shows the *webserver* properties in where the count property has the value 2. It means that two virtual machines will be deployed and attached to the *load-balancer*. Moreover, the rule element is a *listener* which checks the connection requests to the *load-balancer* through Port 80. Finally, the *health-probe* element validates that virtual machines attached to *load-balancer* are available.

Solution Req. 2: Fig. 8 shows a *security-group* which performs like a firewall. It has an outbound rule called *all*, which enable all outgoing connections of virtual machines. Moreover, the *security-group* has two inbound rules (i) *SSH* rule (port 22) allows establishing connections among virtual machine, and (ii) *HTTP* rule (port 80) allows establishing connections with the software application.

Solution Req. 3: Fig. 8 shows a *jumpbox* server, which has connected a public IP (*jumpbox-ip*) to enable connections with servers that are outside the cloud provider. Moreover, the *jumpbox* server performs like a bridge allowing connections between virtual machines attached to the *load-balancer* and servers located outside the cloud provider.

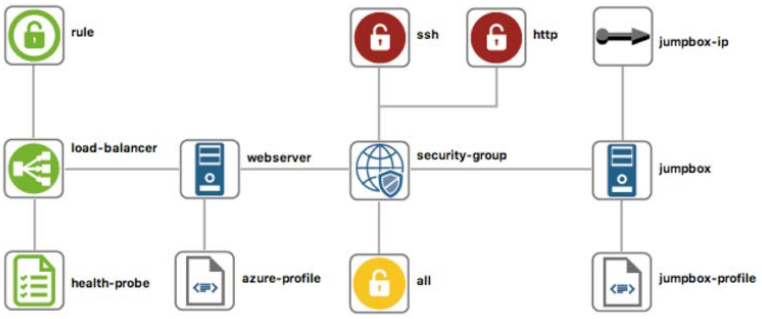

Fig. 8. Load balancer with a jumpbox server.

Following, we explain how to provide a solution for the multi-cloud requirements:

Solution Req. 4: ARGON provides support for modeling a platform-independent model (i.e., Infrastructure Model) and through model-to-model transformation obtain a platform-specific model (i.e., Amazon Model). However, because cloud providers do not offer the same type of infrastructure is necessary to provide extra information such as virtual machine properties. On the one hand, the *jumpbox* server provides information for Amazon Web Services, for instance, image property (see **Fig. 9a**) has the value ami-dc2d10a6 that means an image of a Linux Ubuntu Server 16.04-LTS. On the other hand, jumpbox-profile gives information for Microsoft Azure, for instance, offer, publisher, and SKU properties (see **Fig. 9b**) have the values Ubuntu Server, Canonical, and 16.04-LTS respectively. As a result, the Infrastructure Model (see **Fig. 8**) is capable of being deployed in multi-cloud.

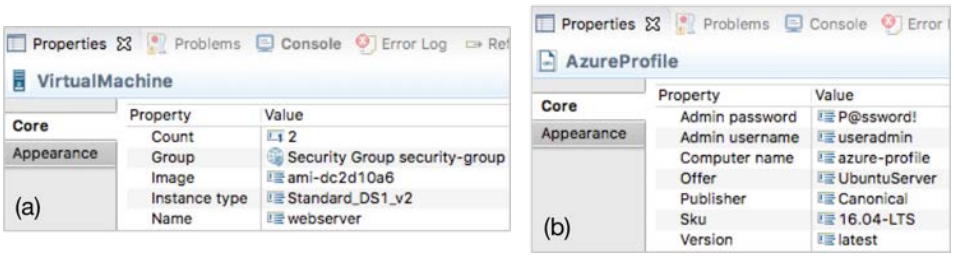

Fig. 9. Virtual machine and cloud profile properties.

Solution Req. 5: ARGON gives support in a migration process based on model-driven techniques. **Fig. 10** presents in a hierarchical tree view the matching among infrastructure elements of the Amazon Model (see **Fig. 10a**) and the Azure Model (see **Fig. 10b**). Following, we explain the main model-to-model transformations: (i) the infrastructure *Service* called load-balancer is a container in the Amazon Model (group A), which is matched to *Resource Group* container, and a *Virtual Network* with its *Subnet* in the Azure Model (group A'). (ii) the *Load Balancer*, *Listener*, and *Health Check* in the Amazon Model (group B) are matching with *Load Balancer*, *Load Balancing Rule*, and *Heath Probe* in the Azure Model (group B'). Also, it is necessary assigned a *Public IP* address for the *Load Balancer* to enable connection requests outside of the Azure platform. (iii) the *Virtual Machines* called webserver and its *Azure Profiles* in the Amazon Model (group C) are matching with *Virtual Machines* and its *Network Interfaces* in the Azure Model (group C'). Also, it is necessary to provide an *Availability Zone* to the *Virtual Machines* in the Azure platform to isolate hardware nodes in a cluster. (iv) the *Virtual Machine* called jumpbox and its *Azure Profile* in the Amazon Model (group D) are matching with a *Virtual Machine* and its *Network Interface* in the Azure Model (group D'). (v) the rest of the infrastructure elements are matching one by one between the Amazon Model and the Azure Model.

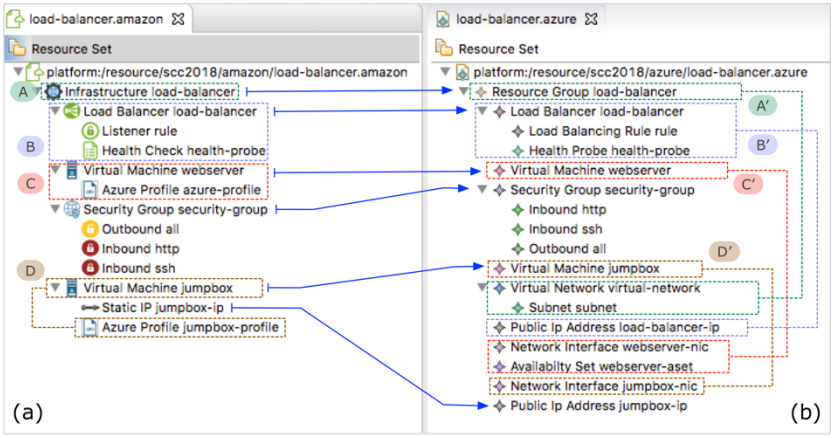

Fig. 10. Elements are matched between load balancer models.

6. Conclusion

In this paper, we have presented an extension of ARGON to demonstrate the feasibility of a multi-cloud infrastructure modeling and a flexible migration approach. The purpose is to provide a novel approach to managing the infrastructure as code in multiple clouds based on model-based techniques. Finally, we believe that our work is aligned with current trends in cloud development and in particular to DevOps as a way to abstract and automate the cloud provisioning of resources given the increasing demand of new applications and the need to reduce the time-to-market.

As future work, we plan to run experiments with practitioners and students with experience in provisioning resources in the cloud to identify new needs and to improve the user experience when dealing with multi-cloud infrastructure provisioning.

Acknowledgements

This research is supported by the MINECO within the Value@Cloud (TIN2013-46300-R) and Adapt@Cloud (TIN2017-84550-R) projects, and SENESCYT.

References

1. Brikman, Y.: Terraform: Up and Running. O'Reilly Media (2017)
2. Casola, V., De Benedictis, A., Rak, M., Villano, U., Rios, E., Rego, A., Capone, G.: MUSA deployer: Deployment of multi-cloud applications. In: Proceedings - IEEE 26th Int. Conf. on Enabling Technologies, WETICE. pp. 107–112. IEEE (2017)
3. Chen, W., Liang, C., Wan, Y., Gao, C., Wu, G., Wei, J., Huang, T.: MORE: A model-driven operation service for cloud-based IT systems. In: Proceedings - IEEE 13th International Conference on Services Computing, SCC. pp. 633–640. IEEE (2016)
4. Ferry, N., Rossini, A.: CloudMF: Model-Driven Management of Multi-Cloud Applications. ACM Trans. Internet Technol. 18 (2), 16–24 (2018)
5. Grozev, N., Buyya, R.: Multi-Cloud Provisioning and Load Distribution for Three-Tier Applications. ACM Trans. Auton. Adapt. Syst. (2014)
6. Humble, J., Farley, D.: Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation. Addison-Wesley Professional (2010)
7. Jouault, F., Allilaire, F., Bézivin, J., Kurtev, I.: ATL: A model transformation tool. Sci. Comput. Program. 72 (1–2), 31–39 (2008)
8. Kolovos, D.S., García-Domínguez, A., Rose, L.M., Paige, R.F.: Eugenia: towards disciplined and automated development of GMF-based graphical model editors. Softw. Syst. Model. 16 (1), 229–255 (2015)
9. Morris, K.: Infrastructure As Code: Managing Servers in the Cloud. O'Reilly (2016)
10. Nitto, E. Di, Matthews, P., Petcu, D., Solberg, A.: Model-Driven Development and Operation of Multi-Cloud Applications. Springer Inter. Publishing, Cham (2017)
11. Rossini, A.: Cloud Application Modelling and Execution Language (CAMEL) and the PaaSage Workflow. In: Proceedings - European Conference on Service-Oriented and Cloud Computing, ESOCC. pp. 437–439. Springer Verlag, Italy (2016)
12. Sandobalin, J., Insfran, E., Abrahao, S.: An Infrastructure Modelling Tool for Cloud Provisioning. In: Proceedings - IEEE 14th International Conference on Services Computing, SCC. pp. 354–361. , Hawai (2017)
13. Steinberg, D., Budinsky, F., Merks, E., Paternostro, M.: EMF: Eclipse Modeling Framework. (2008)
14. Wettinger, J., Breitenbücher, U., Kopp, O., Leymann, F.: Streamlining DevOps automation for Cloud applications using TOSCA as standardized metamodel. Futur. Gener. Comput. Syst. Volume 56 317–332 (2015)
15. Acceleo, https://www.eclipse.org/acceleo/, Accessed: April 22, 2018
16. CloudFormation, https://aws.amazon.com/cloudformation/, Accessed: April 21, 2018
17. OpsWorks, https://aws.amazon.com/opsworks/, Accessed: April 21, 2018
18. Graphical Modeling Framework (GMF) Tooling, https://www.eclipse.org/gmf-tooling/, Accessed: April 22, 2018